Computer Vision Notes

Edge Detection

Faisal Z. Qureshi
http://vclab.science.uoit.ca

Faculty of Science
Ontario Tech University

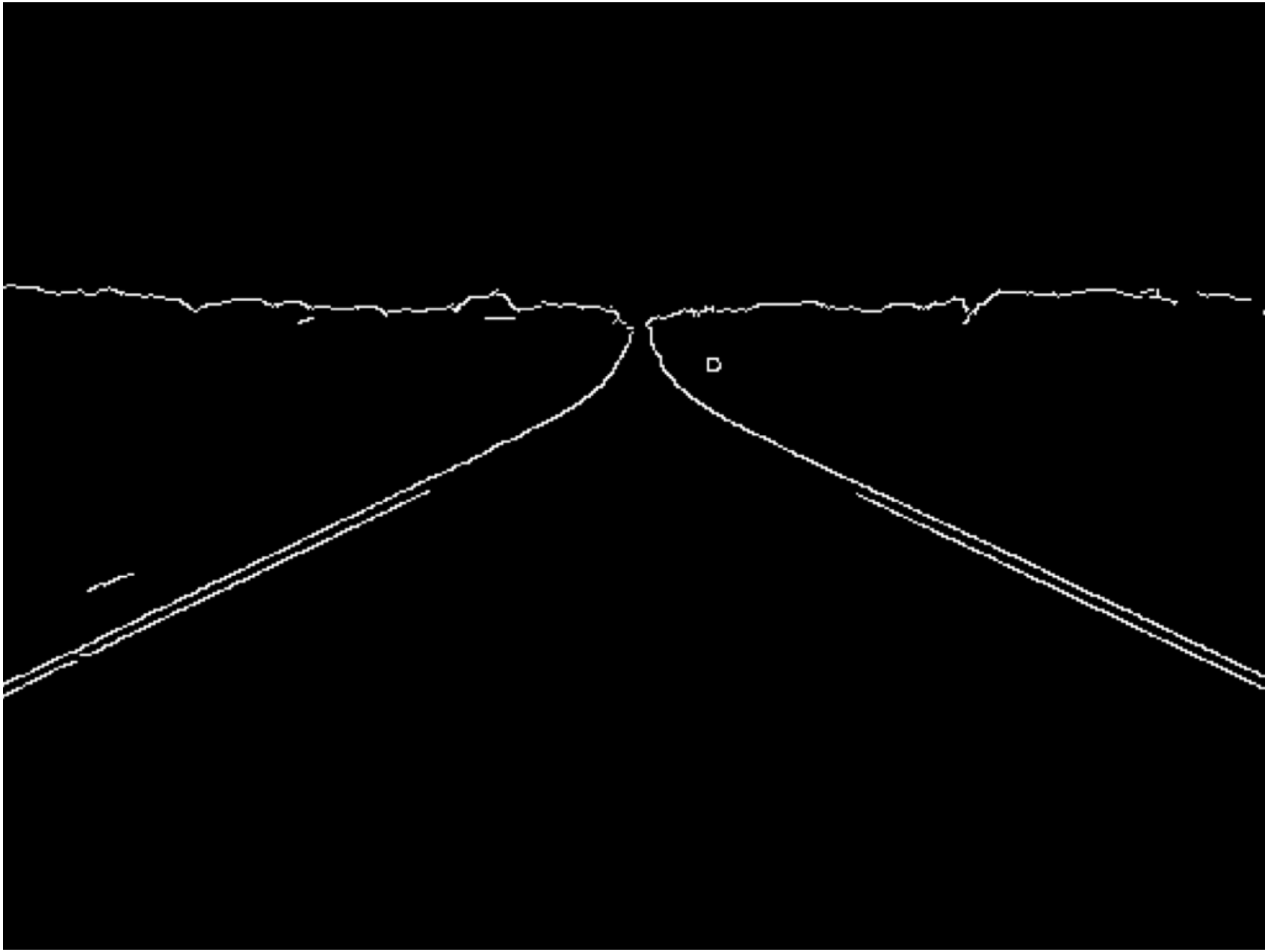
February 8, 2025

Copyright information and license

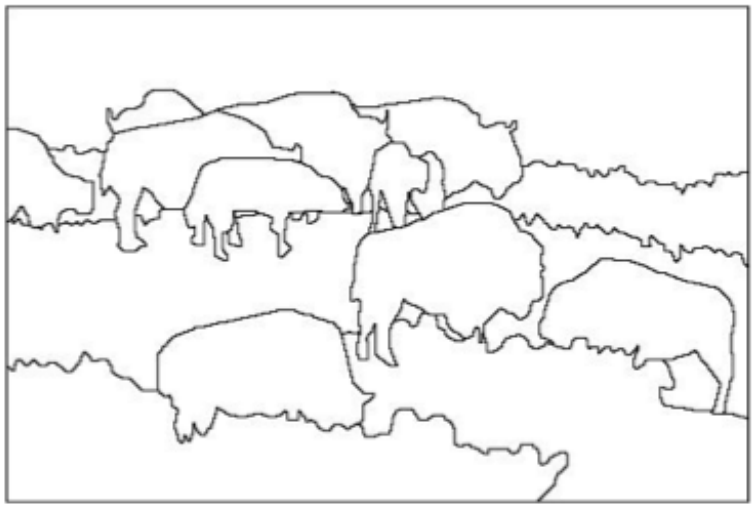
Acknowledgements

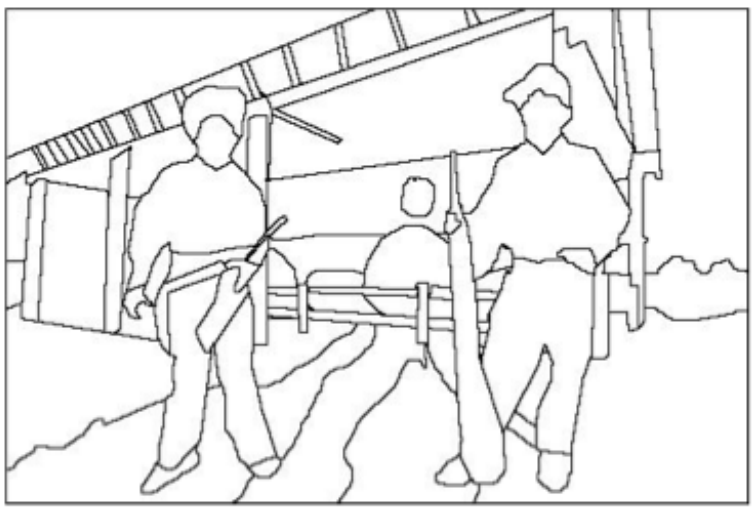
These slides include material from others, including D. Lowe, S. Seitz, K. Grauman, D. Forsyth, and others.

Edge detection



Edge detection





Edge detection

- Identify sudden changes (discontinuties) in an image
- Edges encode semantic and shape information
- Edges are more compact then pixels

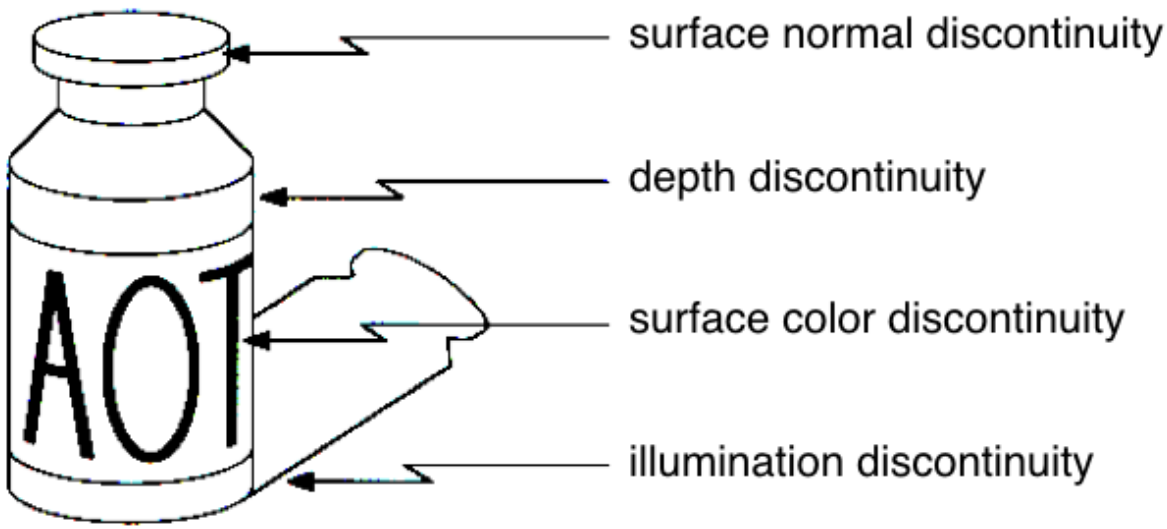
An artist's line drawing

[Source: D. Lowe]

Origin of Edges

Edges are caused by a variety of factors:

- Changes in appearance and texture
- Object boundaries and depth discontinuities
- Changes in surface orientation
- Shadows

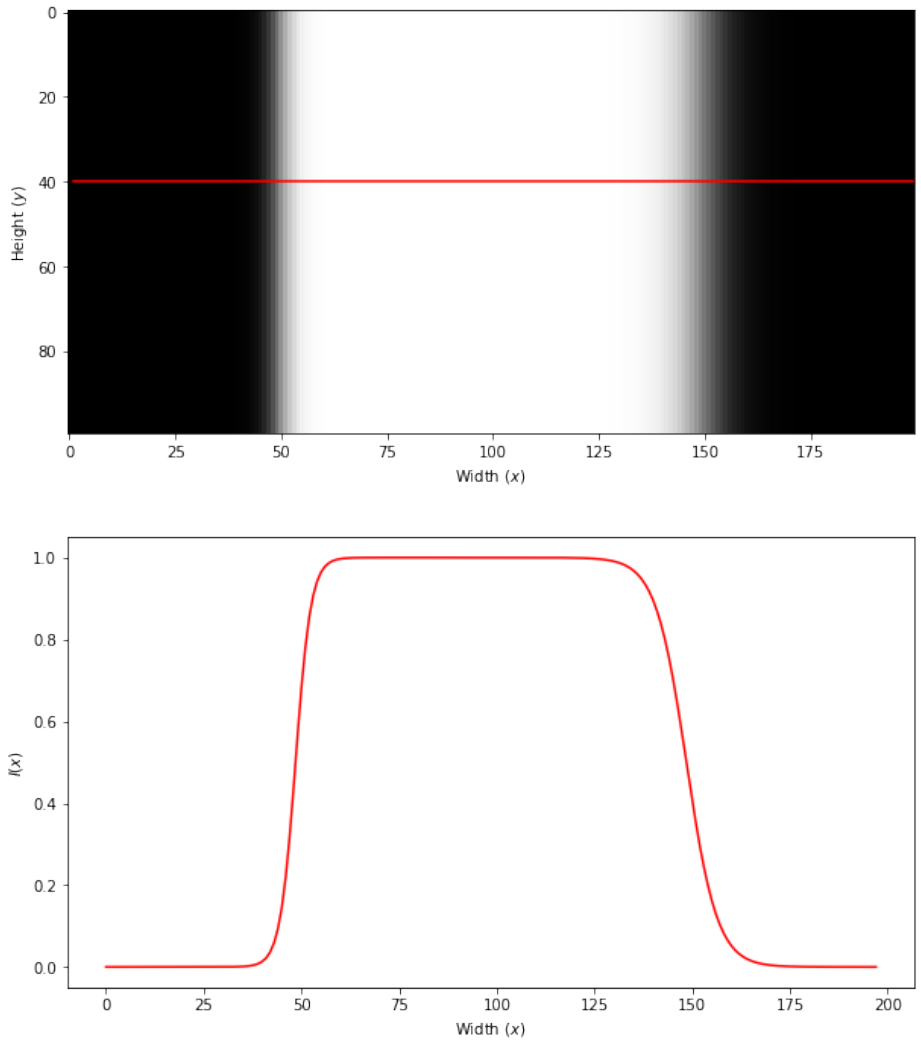


[Source: S. Seitz and K. Grauman]

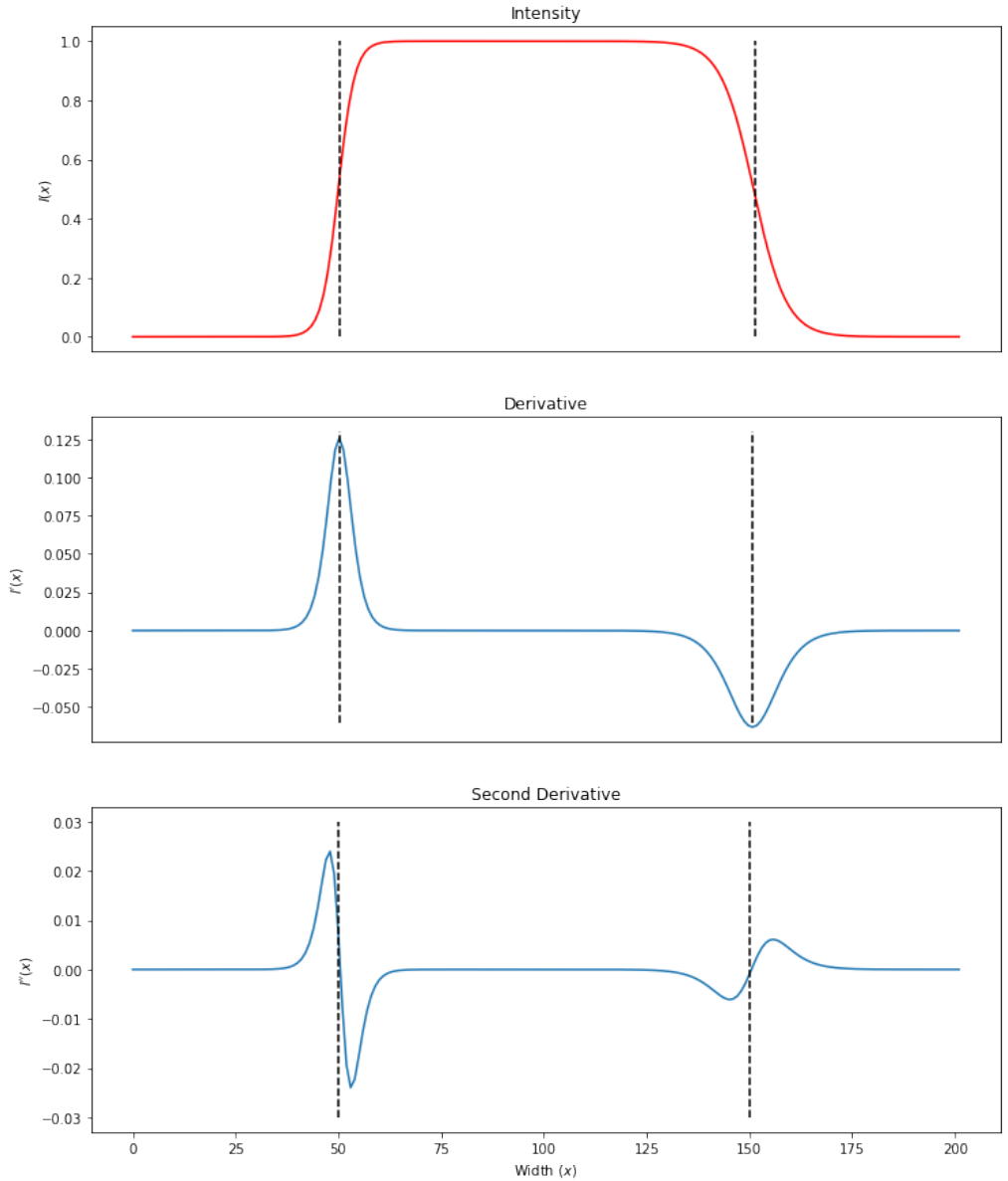
Uses of edge detection

- Object recognition
- Scene understanding
- 3D scene reconstruction

Edges as changes in image intensity



Edges as changes in image intensity



Computing image derivatives

Option 1: Reconstruct a continuous function f, then compute partial derivatives as follows

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x,y)}{\epsilon}$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{\epsilon \to 0} \frac{f(x, y+\epsilon) - f(x,y)}{\epsilon}$$

Option 2: We can use finite difference approximation to estimate image derivatives

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f[x+1, y] - f[x,y]}{1}$$

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f[x, y+1] - f[x,y]}{1}$$

Computing image derivatives using convolutions

We can compute image derivatives using convolution.

Kernel for computing image derivative w.r.t. x (width)

$$H_x = \begin{bmatrix} 1 & -1 \end{bmatrix}$$

Kernel for computing image derivative w.r.t. y (height)

$$H_y = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Finite difference filters

- Sobel filters

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } H_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

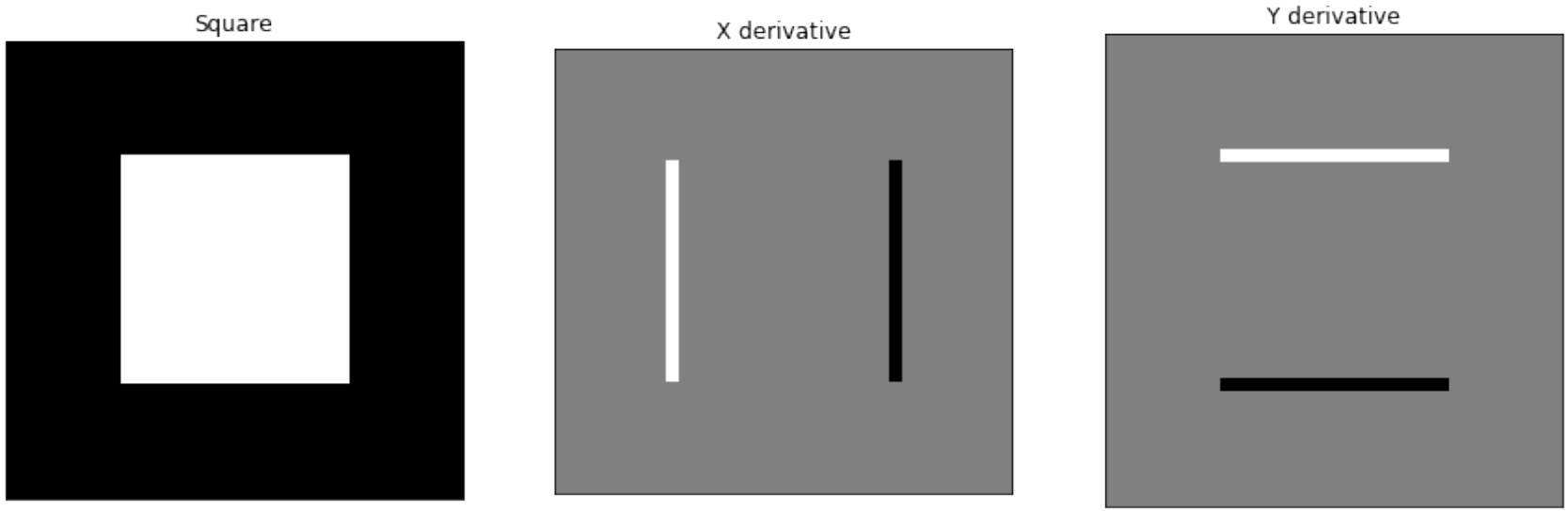
- Prewire

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } H_x = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

- Roberts

$$H_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \text{ and } H_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Image derviatives highlights edge pixels



- X derivative is computed by convolving image with H_x, and Y derivative is computed by convolving image with H_y.
- Pixels sitting on vertical edges are highlighted in X derivative.
- Pixels sitting on horizontal edges are highlighted in Y derivative.

Image gradient

- Image gradient ∇I points to the direction of most rapid change in intensity.

$$\nabla I = \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{bmatrix}^T$$

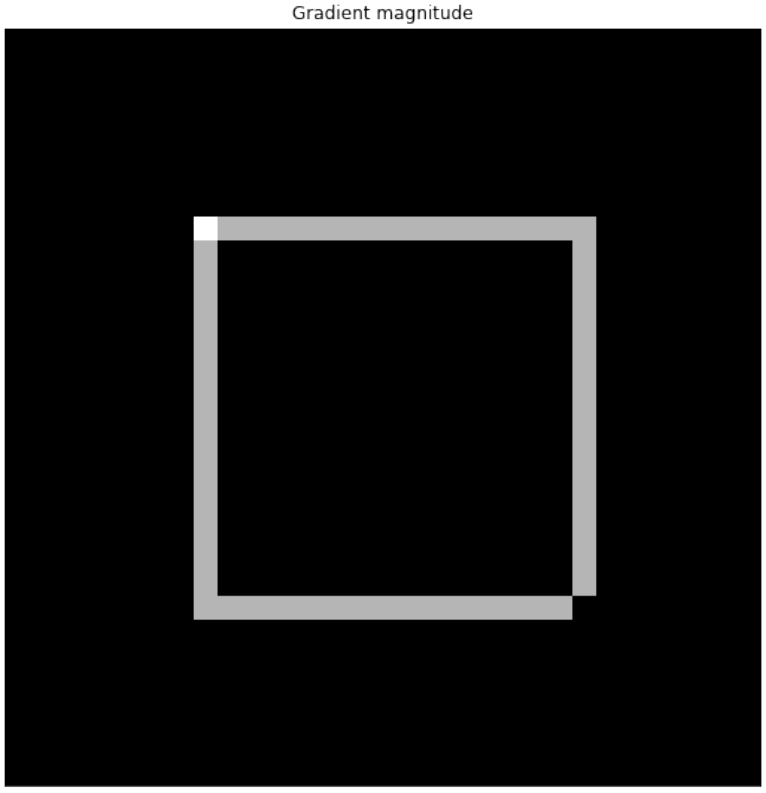
- Gradient magnitude (can be used to identify edge pixels)

$$\|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

- Gradient direction (is perpendicular to the underlying edge)

$$\theta = \tan^{-1}\left(\frac{\partial I}{\partial x} / \frac{\partial I}{\partial y}\right)$$

Image gradient example



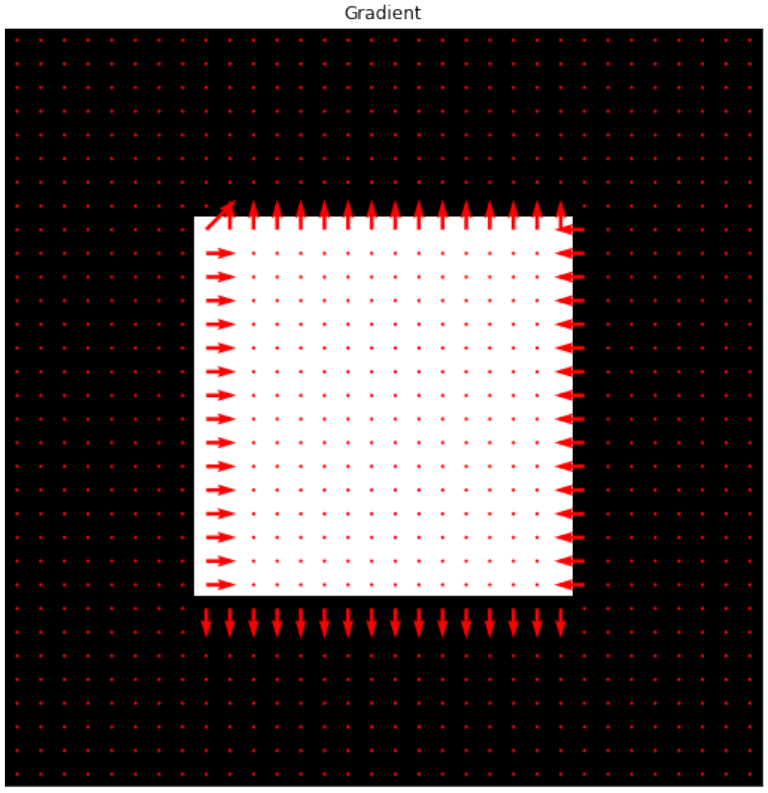
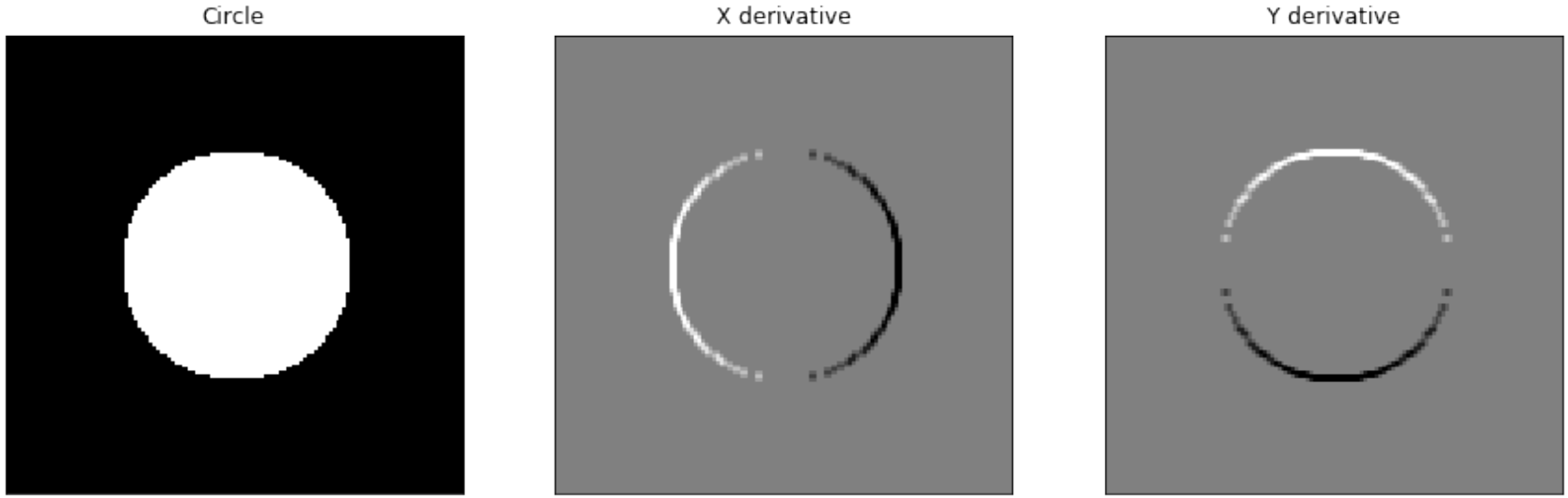
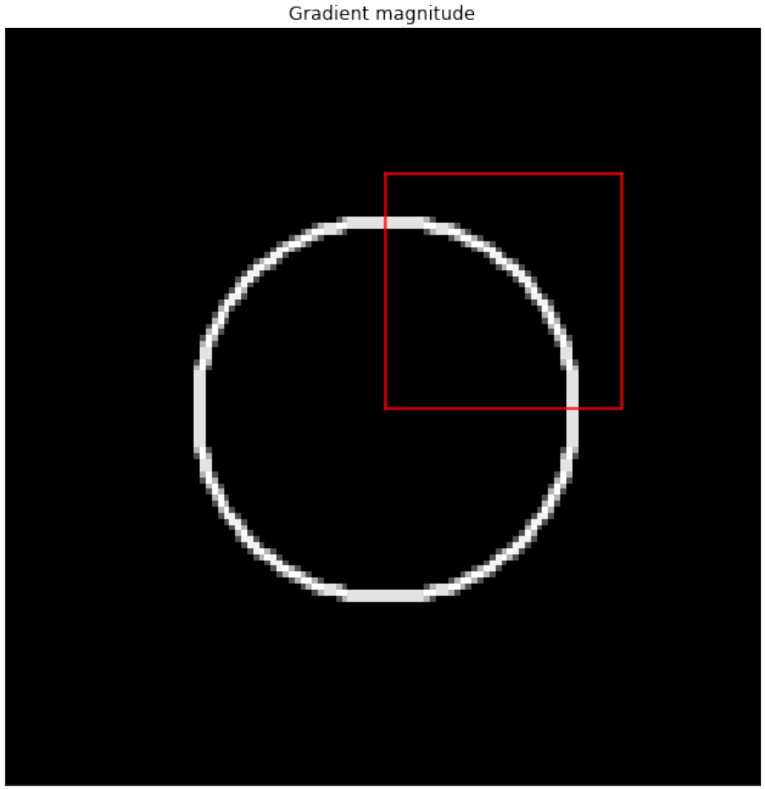


Image derviatives highlights edge pixels



- Using *Sobel filters*
- X derivative is computed by convolving image with H_x, and Y derivative is computed by convolving image with H_y.
- Pixels sitting on vertical edges are highlighted in X derivative.
- Pixels sitting on horizontal edges are highlighted in Y derivative.

Image gradient example



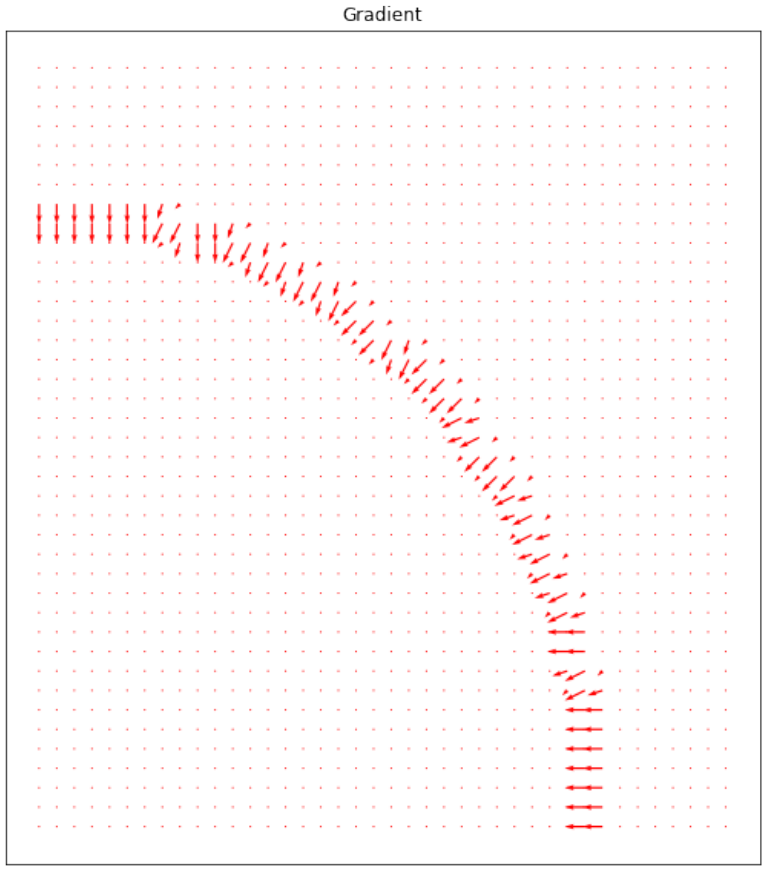


Image gradients capture edge normals in addition to edge strength.

Effect of noise on edge detection

Consider the following 1d image.

$f(x)$

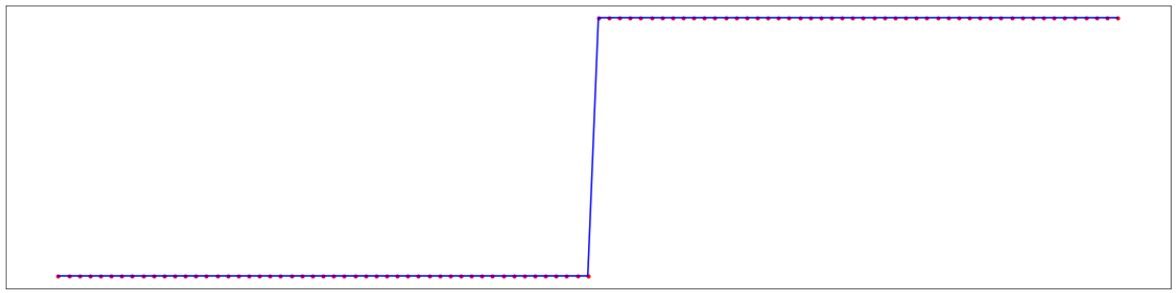
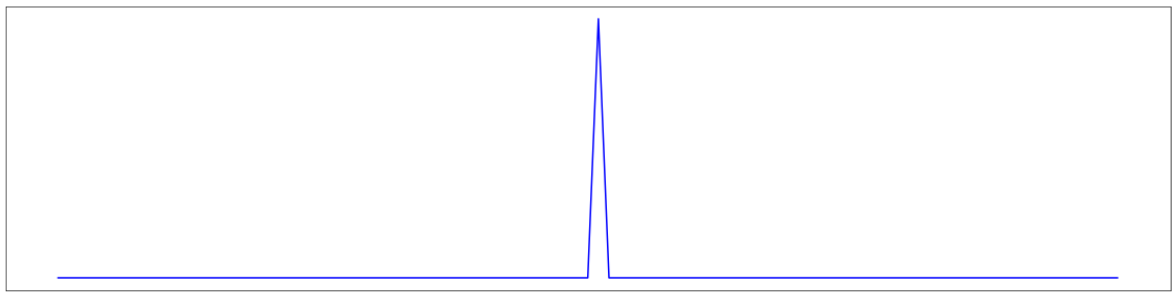


Image derivative successfully localizes the edge seen in this image.

$df(x)/dx$



Effect of noise on edge detection

Now consider the same 1d image, but this time corrupted with some noise. Notice that the edge is still seen in the image.

f

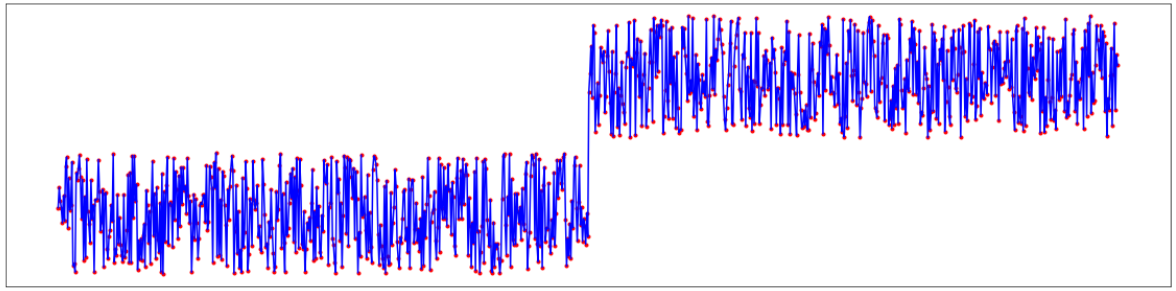


Image derivative, however, is unable to localize the edge in this case.

$\frac{d}{dx} f$

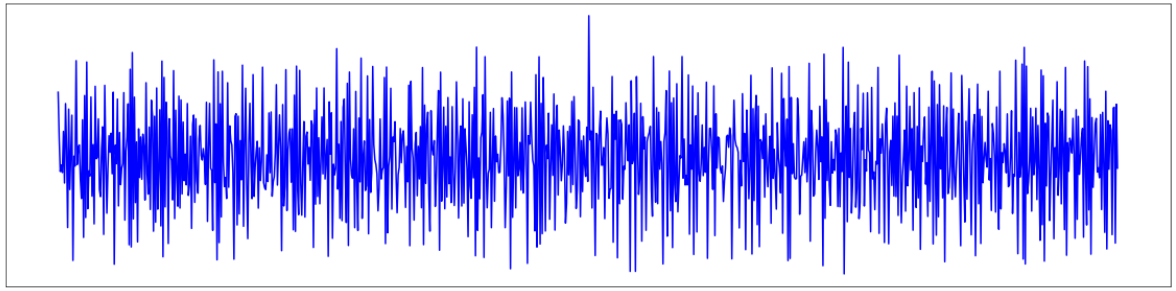
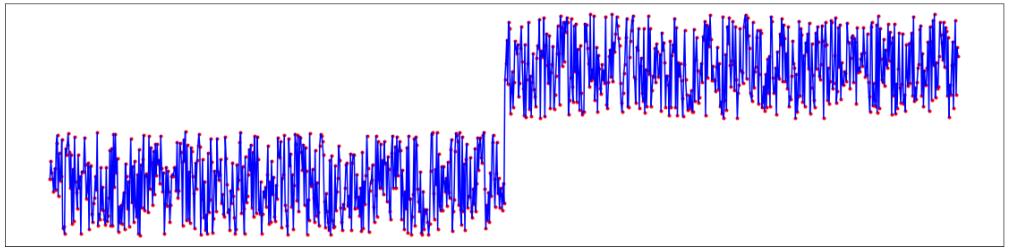


Image derivatives are highly sensitive to the presence of noise in the image. Use smoothing first to get rid of high-frequency components.

Smoothing to reduce the effect of noise

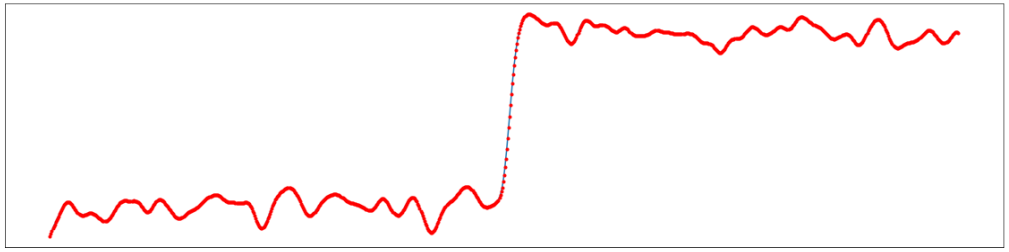
Given noisy image

f



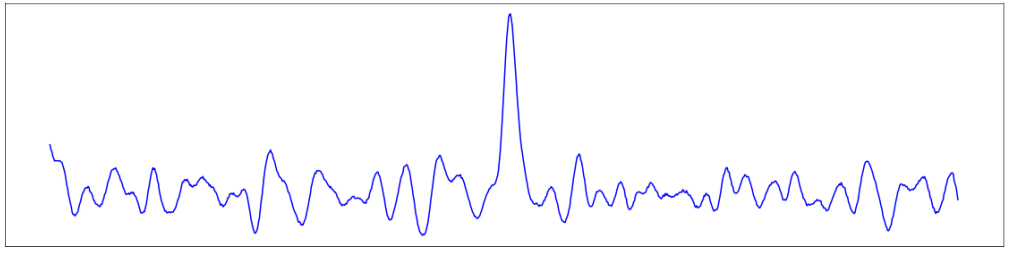
Step 1: Perform Gaussian filtering

$G * f$



Step 2: Compute image derivative

$\frac{d}{dx}(G * f)$



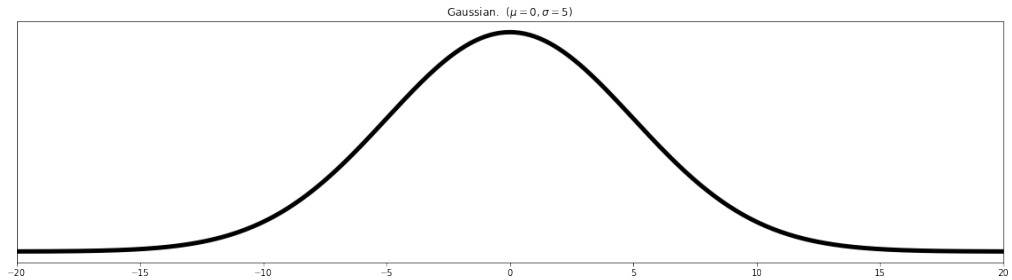
Smoothing to reduce the effect of noise

Recall that convolution (linear filtering) is associative, meaning

$$\frac{d}{dx}(G * f) = \left(\frac{d}{dx}G\right) * f$$

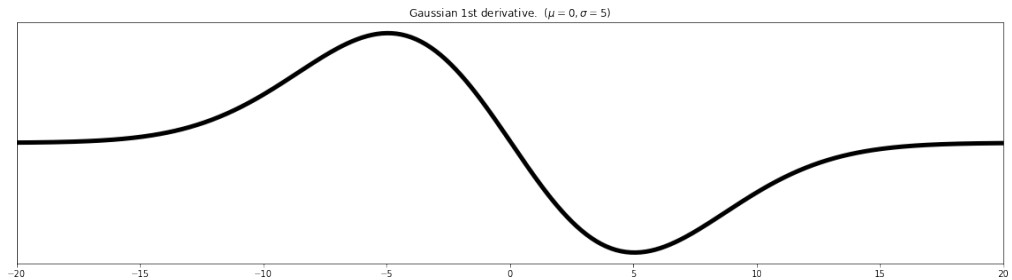
Gaussian

G



First derviative of Gaussian

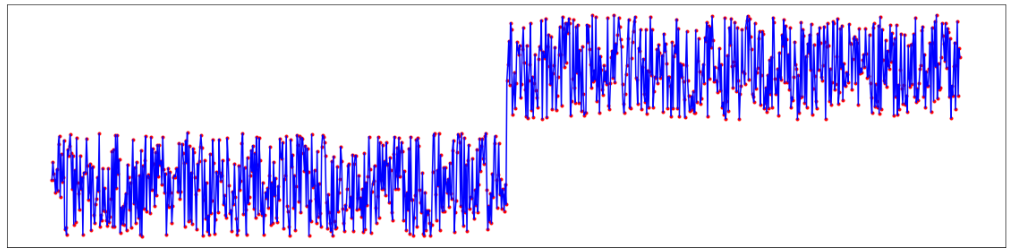
$\frac{d}{dx}G$



Smoothing to reduce the effect of noise

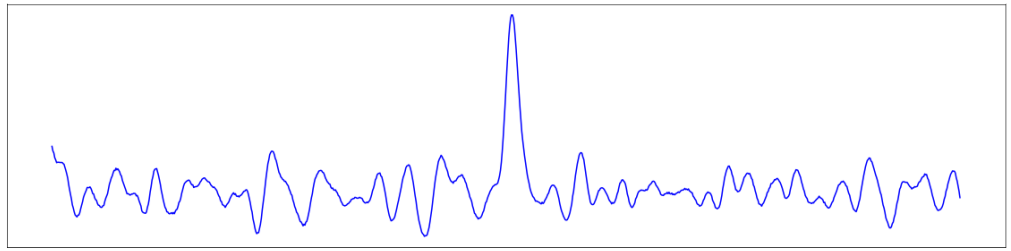
Given noisy image

f



Step 1: Convolve with first derivative of Gaussian

$\left(\frac{d}{dx}G\right) * f$

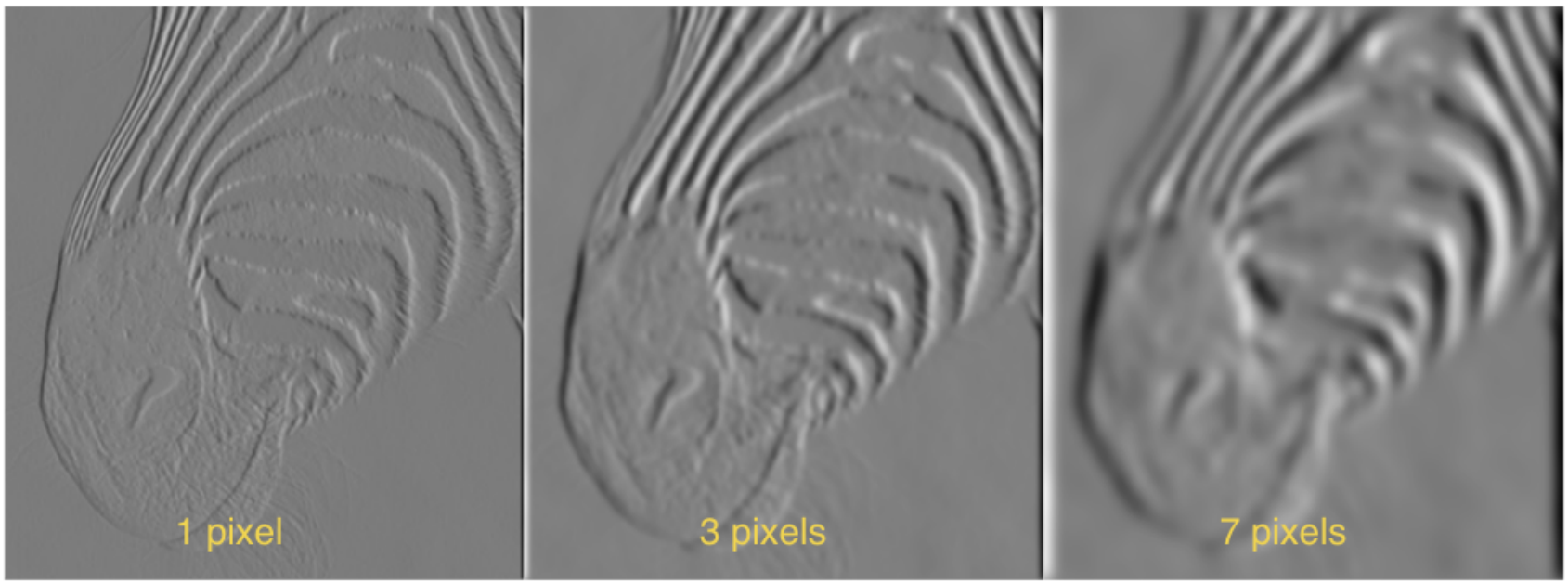


This saves us one convolution operation.

It is a good practice to always smooth an image before computing its gradient.

Image smoothing & edge localization

Smoothed derivative removes noise, but blur the edges, making edge localization challenging.



[Source: D. Forsyth]

Image smoothing & edge localization

- The gradient magnitude is large along a thick 'ridge'. How do we identify (localize) the actual edge pixels?
- How do we link edge pixels to form contours?

[Source: D. Forsyth]

Designing an edge detector

- An edge detector should find all "real" edges, ignoring noise and other artifacts
- An edge detector should have good localization property, i.e., the detected edges should be as close to real edges as possible.

Cues that we can use for developing a useful edge detector

- Intensity, color or texture differences
- Continuity or closure
- High-level or semantic knowledge

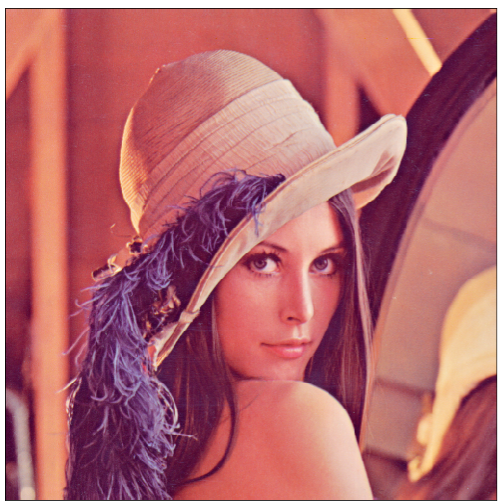
Canny edge detector

Canny, J., *A Computational Approach To Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

- Most widely used edge-detector in computer vision
- Theoretical model: step-edges corrupted by Gaussian noise
- Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of signal-to-noise ratio and localization

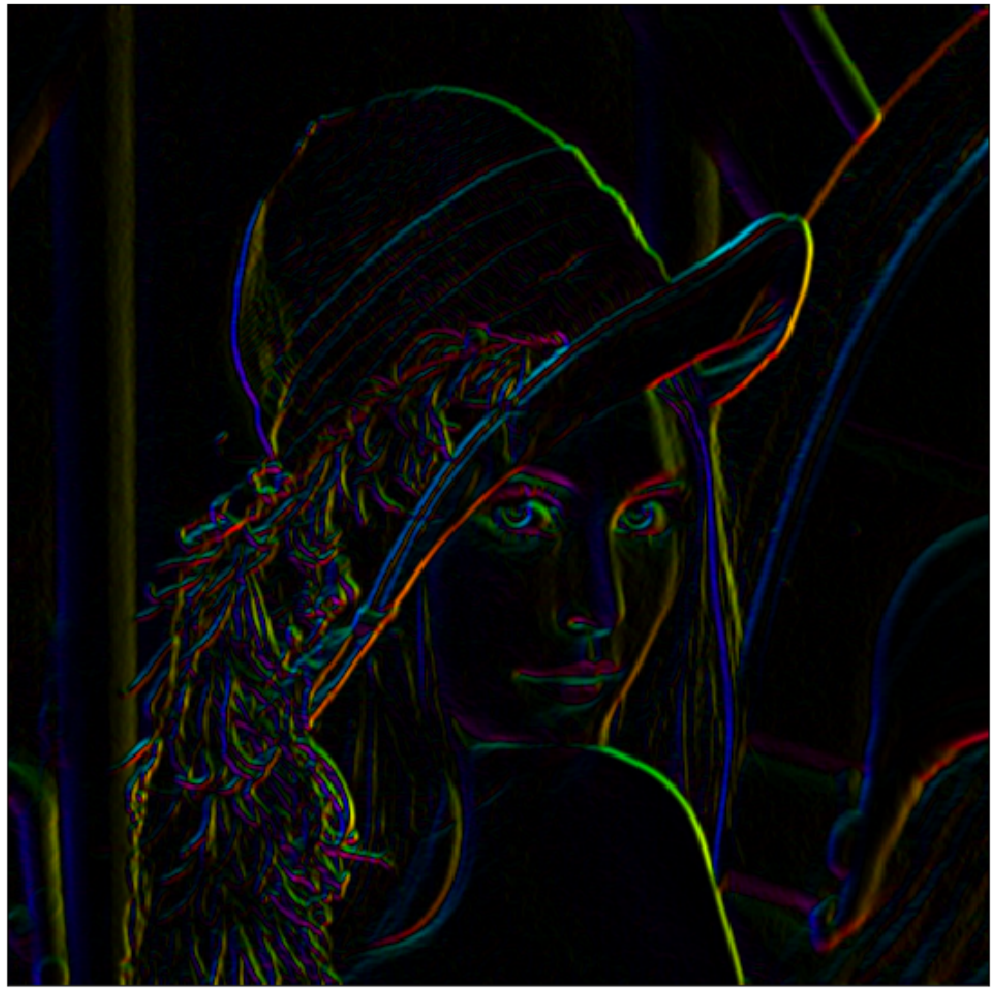
Canny edge detector - Edge pixel identification

Step 1 Identify edge pixels using gradient magnitude



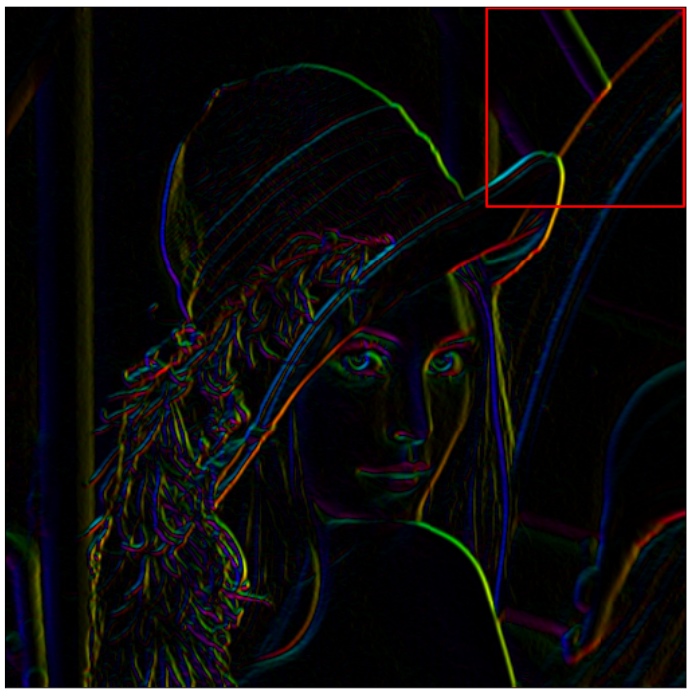
Canny edge detector - Edge orientations

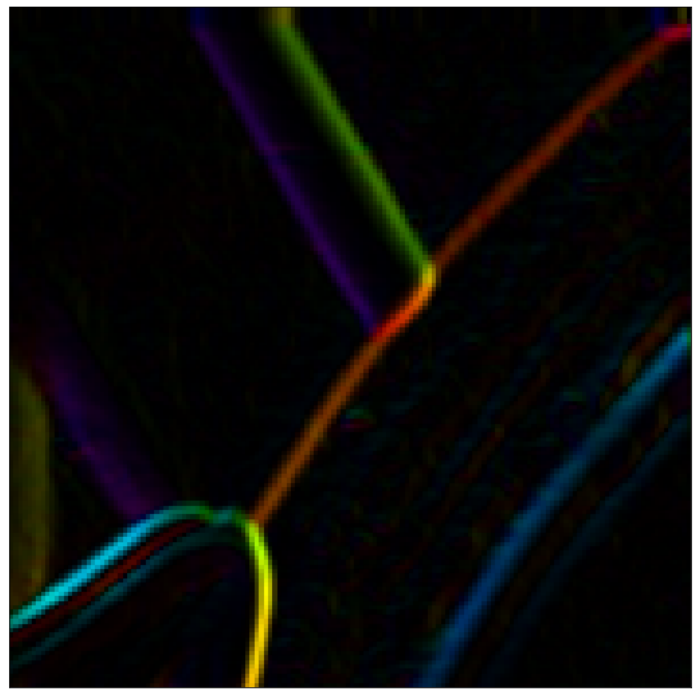
Step 2 Compute edge orientations using image gradient



Canny edge detector - Edge localization

Step 3 In order to achieve good edge localization, we need to thin out the edges.

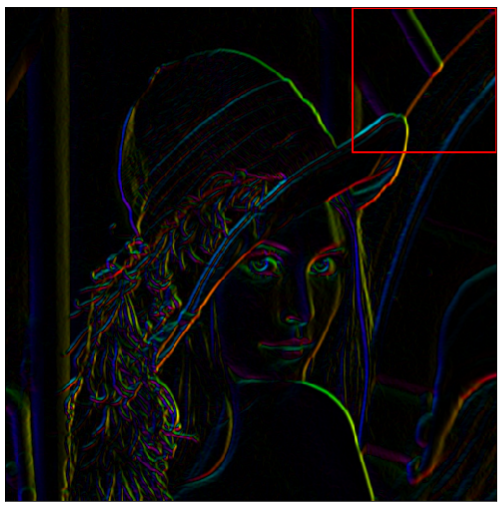


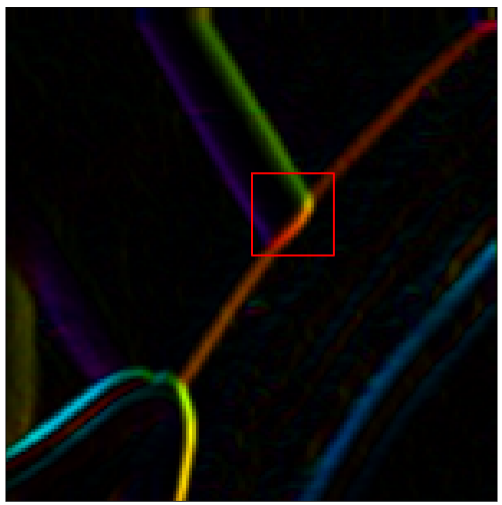


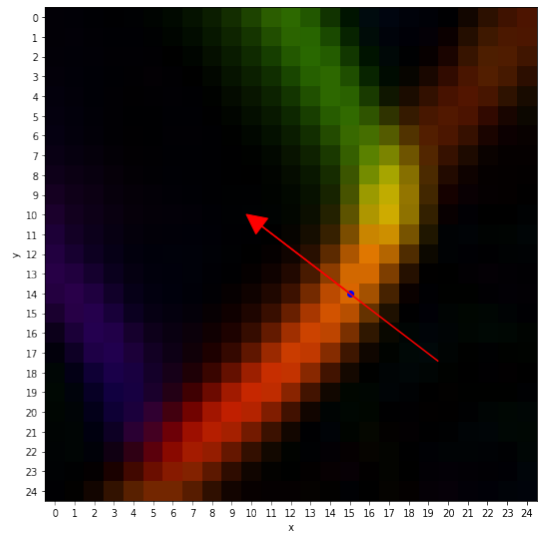
Canny edge detector - Edge localization

Step 3 In order to achieve good edge localization, we need to thin out the edges.

We will use non-maxima-suppression to achieve that. Non-maxima suppression needs to account for the orientation at each location.



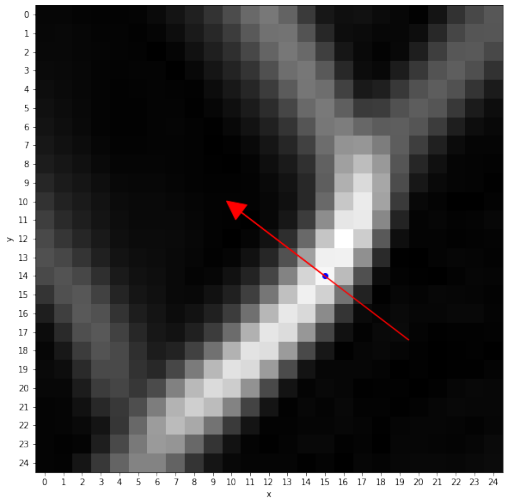


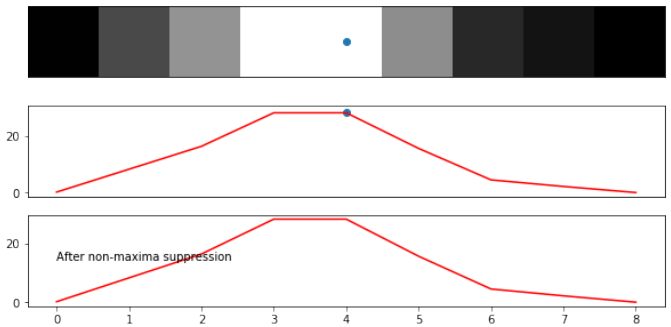


Canny edge detector - Non-Maxima Suppression

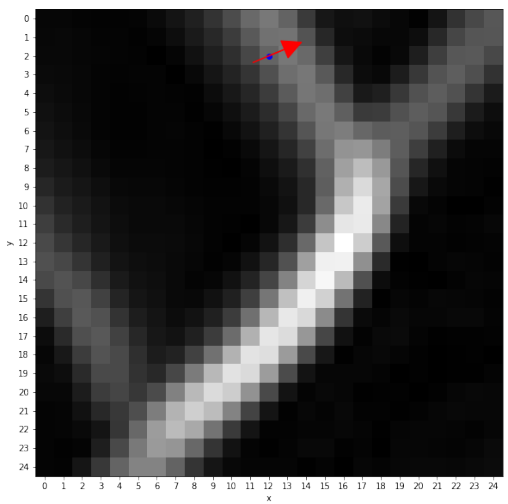
Sample pixel values across the edge (using gradients) and only keep a pixel if it has the maximum value.

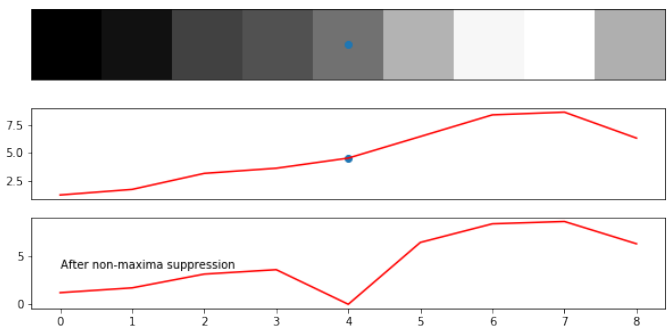
Pixel is kept since it has the largest gradient magnitude across the edge





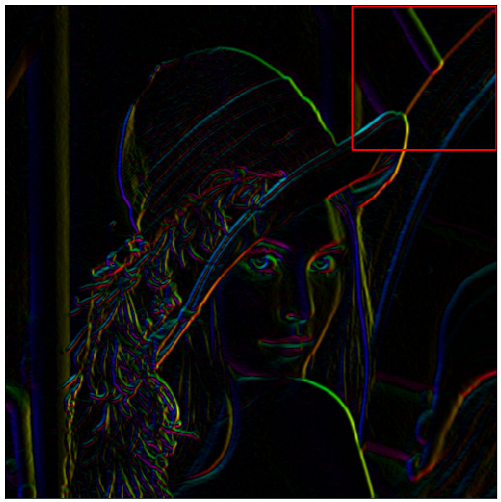
Pixel is discarded since it does not have the largest gradient magnitude across the edge

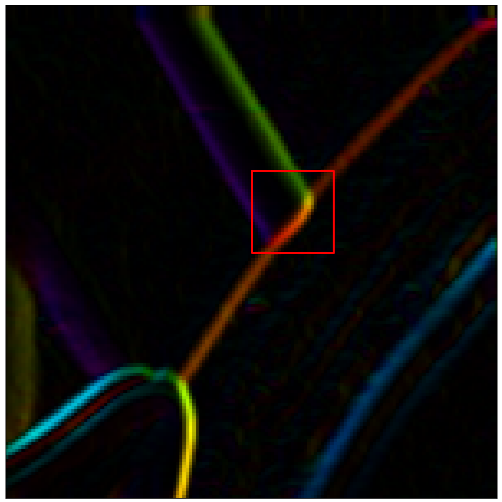


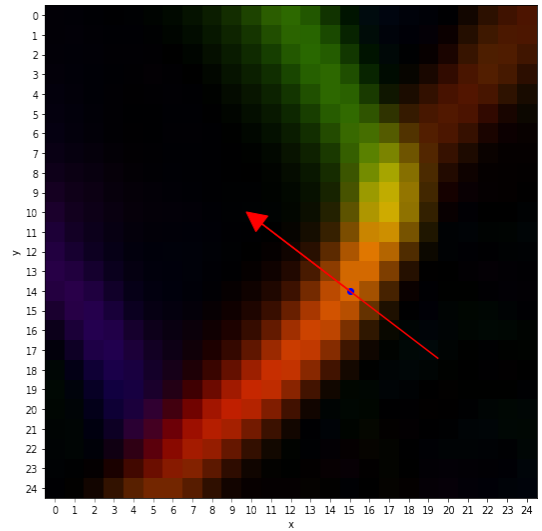


Canny edge detector - Non Maxima Suppression

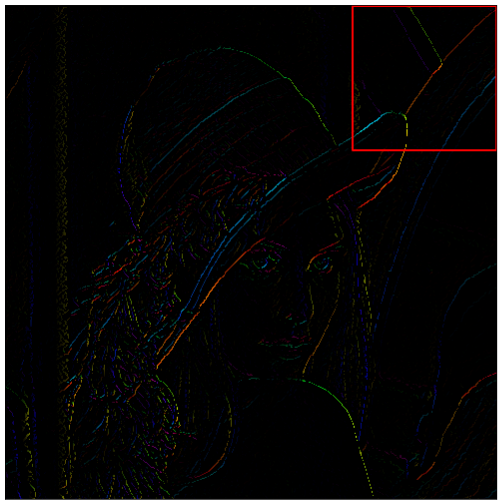
Before

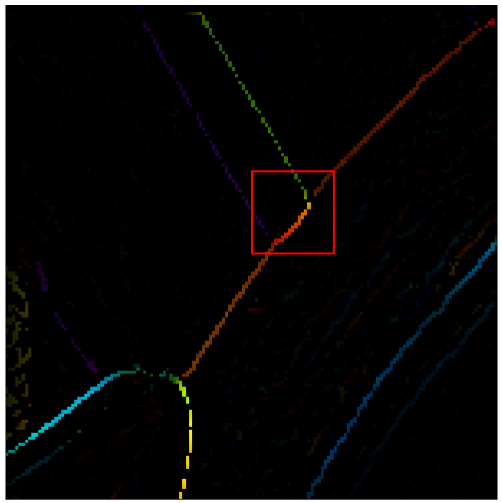


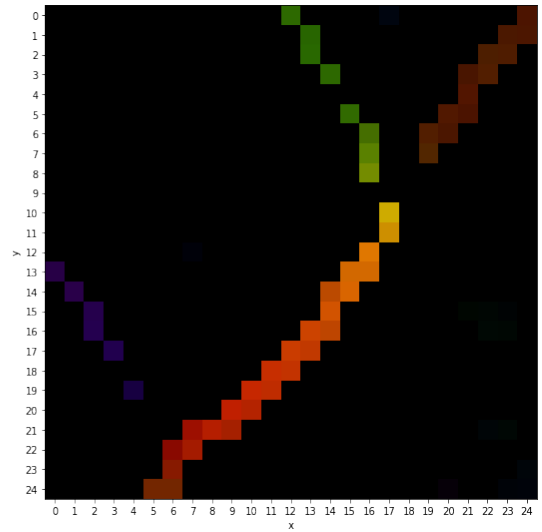




After





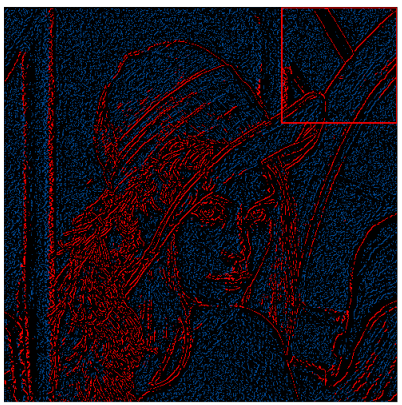


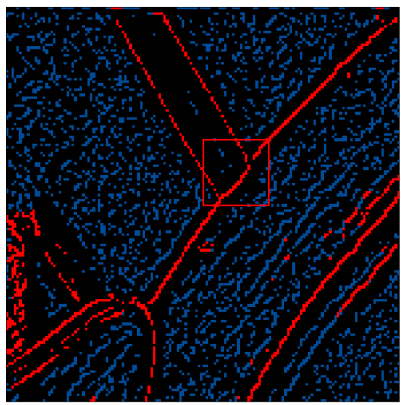
Canny edge detector - Discard non-edge pixels

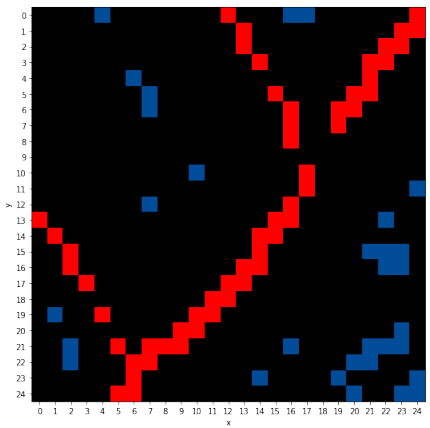
Pick two thresholds t_1 and t_2, where $t_1 < t_2$, and use these to discard non-edge pixels.

$$E(x,y) = \begin{cases} \text{not an edge,} & \text{when } |\nabla I(x,y)| < t_1 \\ \text{weak edge,} & \text{when } t_1 \leq |\nabla I(x,y)| \leq t_2 \\ \text{strong edge,} & \text{when } |\nabla I(x,y)| > t_2 \end{cases}$$

E captures the information about non-edge, weak-edge, and strong-edge pixels.

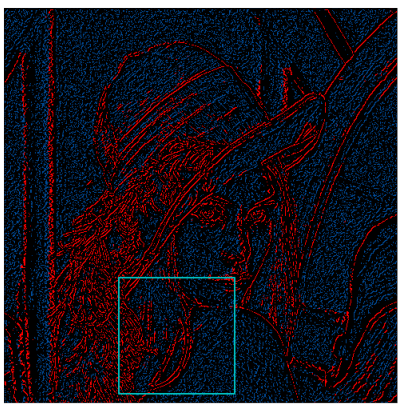


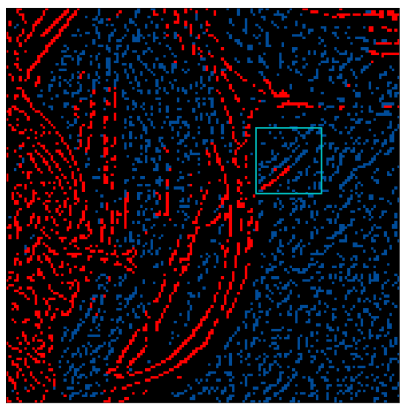


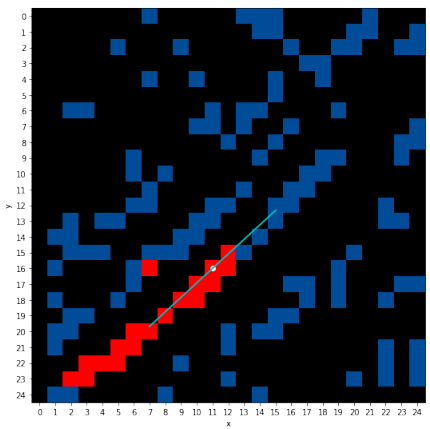


Canny edge detector - Hysteresis

- Start edges at “strong” edge pixels and continue these on “weak” pixels that fall along these edges.
- Use **tangent** at each pixel to identify neighbouring pixels that fall along the edge.

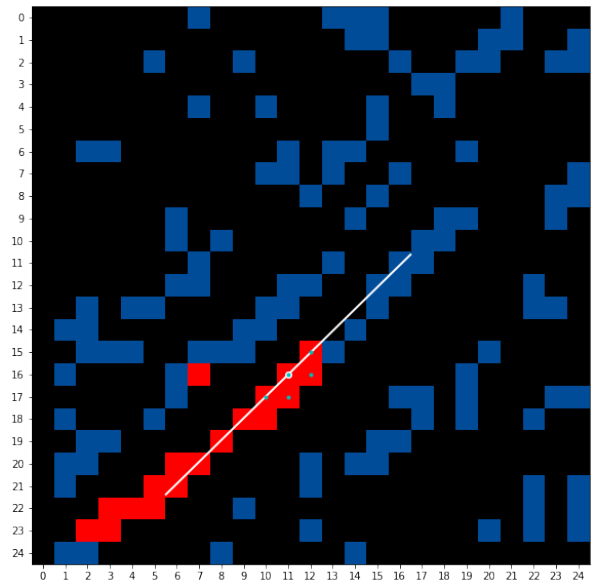


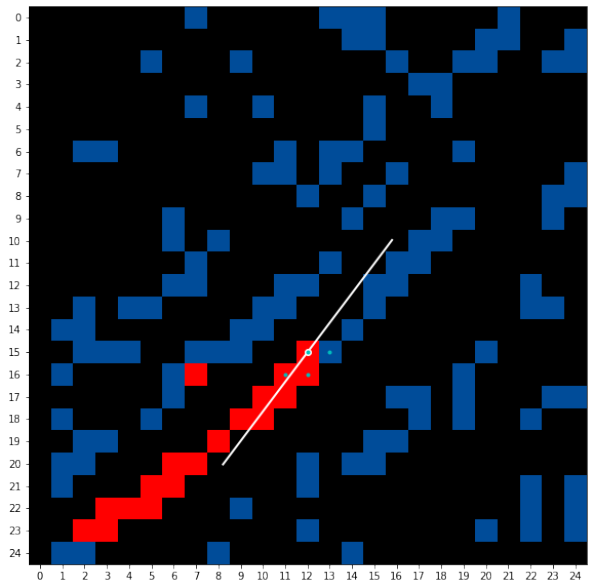




Canny edge detector - Hysteresis

Use edge tangent to identify neighbouring pixels that fall along the same edge.



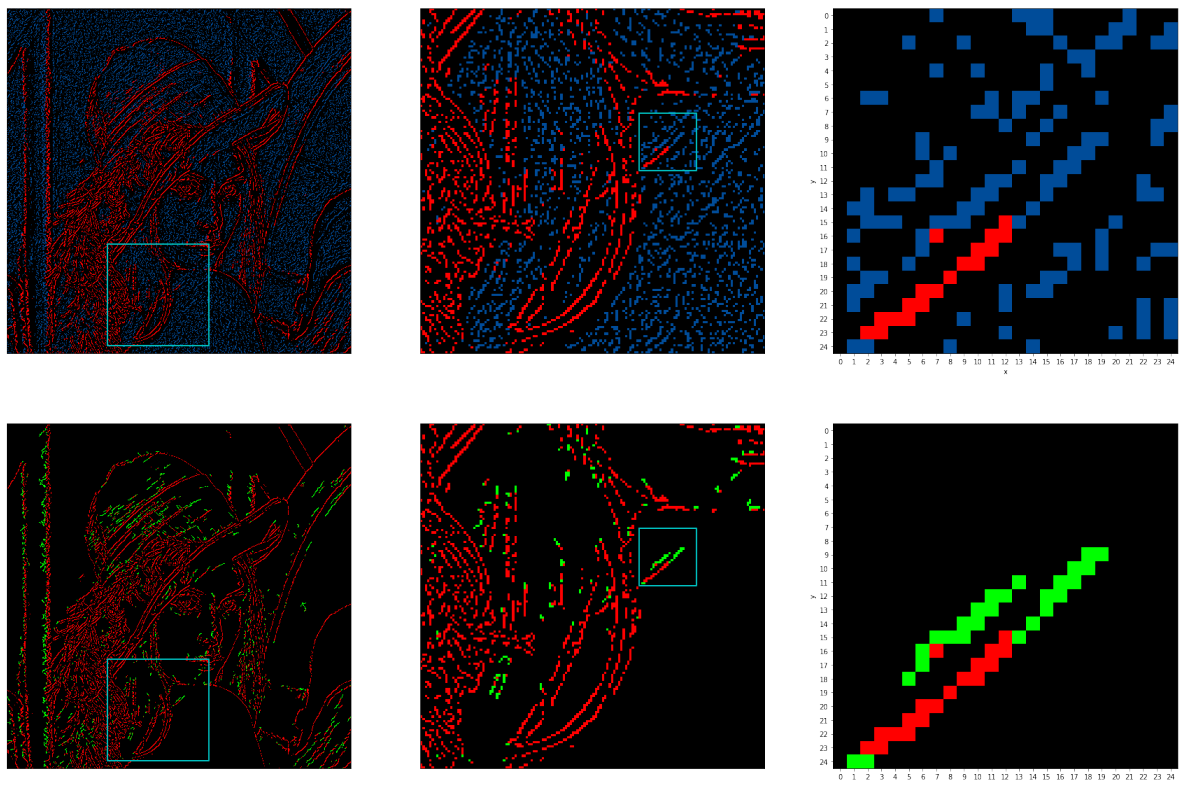


Tangent at each pixel is easily computed by rotating gradient at that pixel by 90 degrees.

Canny edge detector - Hysteresis

Weak pixels that **do not** fall along edges containing strong pixels are discarded.

In the figure below, notice that some blue pixels are turned off (set to black), while others are set to green. Both red and green pixels are edge pixels. Before (top row) and after (bottom row) performing hysteresis based edge identification.

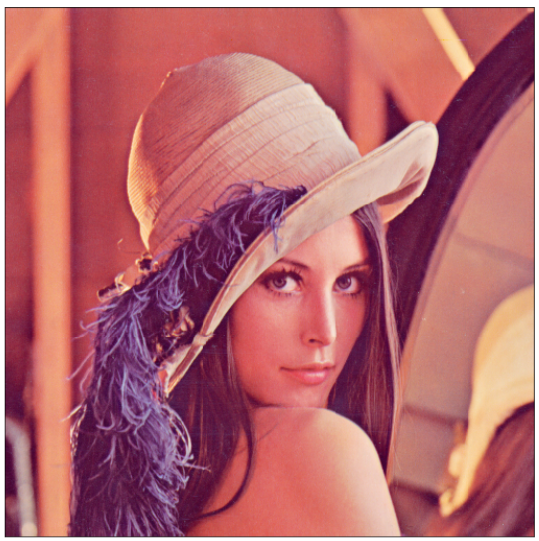


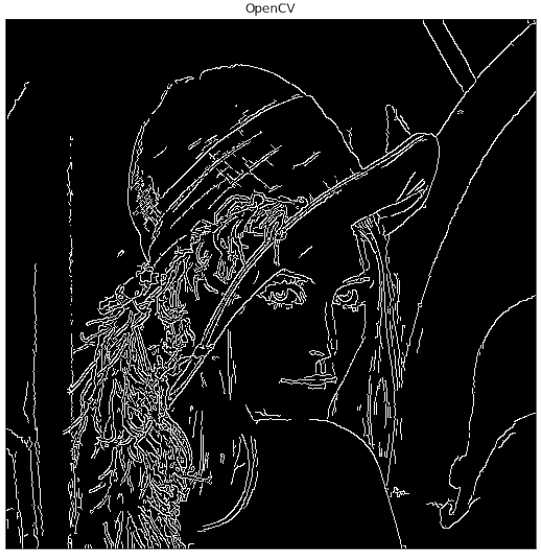
Canny edge detector - Open CV implementation

OpenCV includes an implementation of Canny edge detector.

```
import cv2
import numpy as np

img_bgr = cv2.imread(filename)
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
canny = cv2.Canny(img_gray, threshold1=60, threshold2=120, apertureSize=3, L2gradient=True)
```

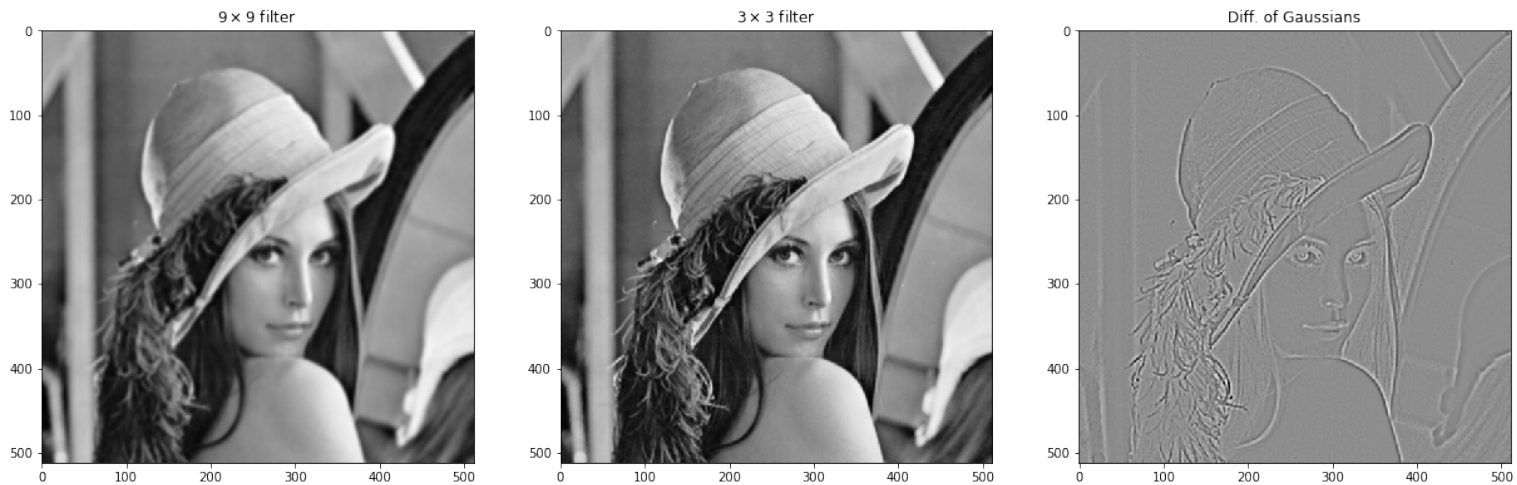




Difference of Gaussian (DoG)

DoG can be used to approximate smoothed gradient of an image as seen below.

$$\frac{\partial}{\partial x}(G_\sigma * I) \approx (G_{\sigma_1} * I) - (G_{\sigma_2} * I)$$



Boundary detection

There has been lot of interest in boundary detection and object segmentation. More on that later.